Data Wrangling with SQL

Good morning!

We start at 10:00

Codd's Normal Forms (1)

The Goal:

1. Eliminate data inconsistencies
2. Minimize storage space as much as possible with respect to 1

→ Constraint-based optimization problem!

First Normal Form (1NF) (1)

All attributes (columns) of a relation (table) should be atomic in value.

- atomic → smallest / non-decomposable
- value → data information

Breach of 1NF

1NF (2)

CUSTOMERS (CustID, CustName, Address)

CustID	CustName	Address
1	JT (Ho)	10 Downing Street, London, UK

Address: street # | street name | City | Country

INF is respected

INF (3)

Addresse

CUSTOMERS (CustID, CustName, StreetH, StreetName, City, Country)

CustID	CustName	St.H	St. Name	City	Country
1	TT (Ho)	10	Downing	London	UK

Second Normal Form (2NF) (1)

For 2NF to apply:

1NF should apply

+ There shall be no functional dependency between part of the key and a non-key attribute

INF → YES
2NF → NO

2NF (2)

Player (Person, Sport, Height, Weight)

Key Key Not-Key Not-Key

Person	Sport	Height	Weight
Sebastian	Run	190	70
Hanna	Swim	170	70
Hanna	Tennis	170	70

Breach of 2NF

1NF → Yes
2NF → Yes

2NF (3)

∴ 2NF respected

Olympic (Person, Sport)

Person	Sport
Sebastien	run
Hanna	swim
Hanna	tennis

Player (Person, Height, Weight)

Person	Height	Weight
Sebastien	190	90
Hanna	170	70

Third Normal Form (3NF) (1)

For 3NF to apply:

2NF should apply

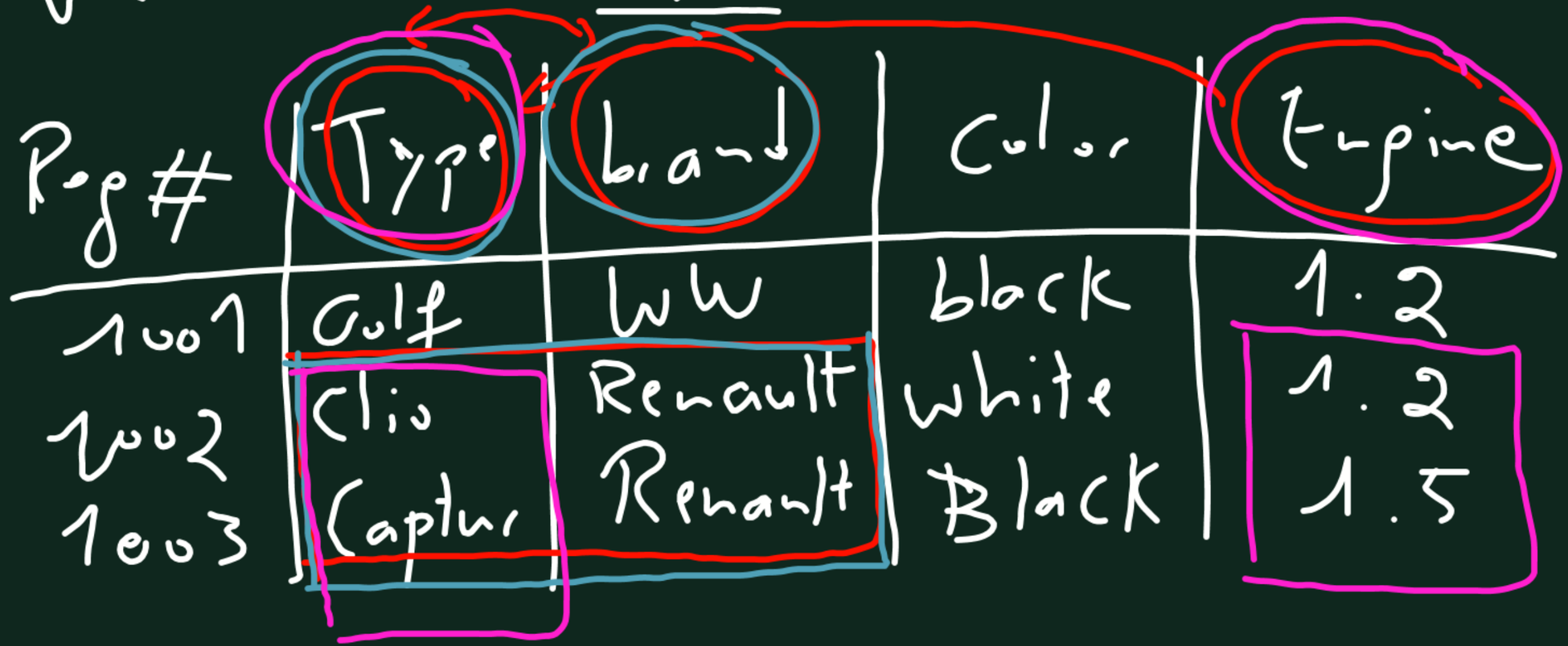
+ There shall be no functional dependency between non-key attributes

1NF → Yes
2NF → Yes
3NF →

3NF (2)

Breach of 3NF

VEHICLE (Reg#, Type, brand, Color, Engine)



Reg#	Type	brand	Color	Engine
1001	Golf	WW	black	1.2
1002	Clio	Renault	white	1.2
1003	Captur	Renault	Black	1.5

1NF → Yes
2NF → Yes
3NF → Yes

3NF (3)

3NF is "respected!"

FK

PK

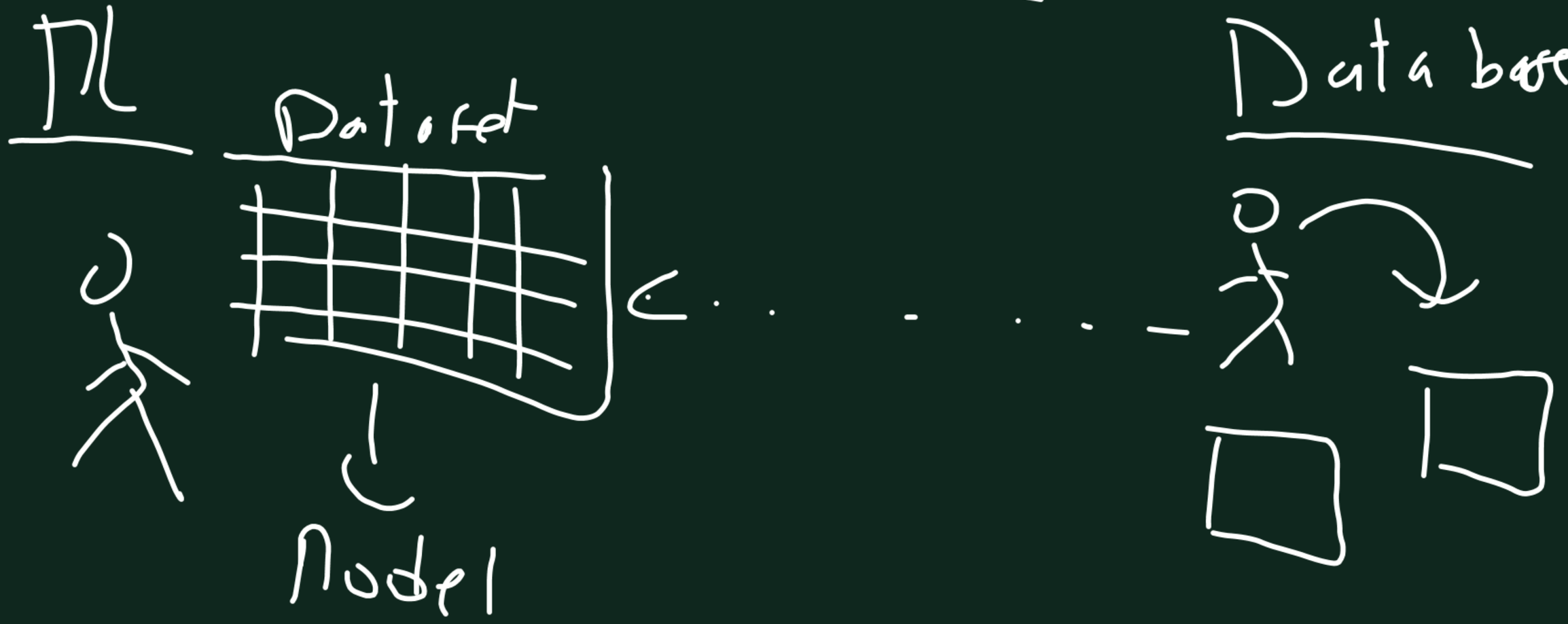
VEHICLE(Reg#, Type, Color)

MODEL(Type, Brand, Engine)

Reg#	Type	Color
1001	Golf	Black
1002	Clio	White
1003	Captur	Black

Type	Brand	Engine
Golf	VW	1.2
Clio	Renault	1.2
Captur	Renault	1.T

Chain of Data



Codd's Normal Forms (2)

Recent findings (in the 70's) showed that while respecting the NF, roughly 5% of dbs are still inconsistent

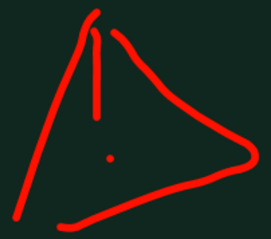
Boyce - Codd Normal Form (BCNF)

There shall be no other functional dependency aside from key and non-key attributes

excludes
key - key
partial key - key, etc.

Break

Back at 11:50



Normal Forms: A Theoretical Reference

T(A, B, C, D)

1NF: A, B, C, D are atomic

2NF: A, B → C
A, B → D

Not A → C
Not A → D
NoT B → C
NoT B → D

3NF:

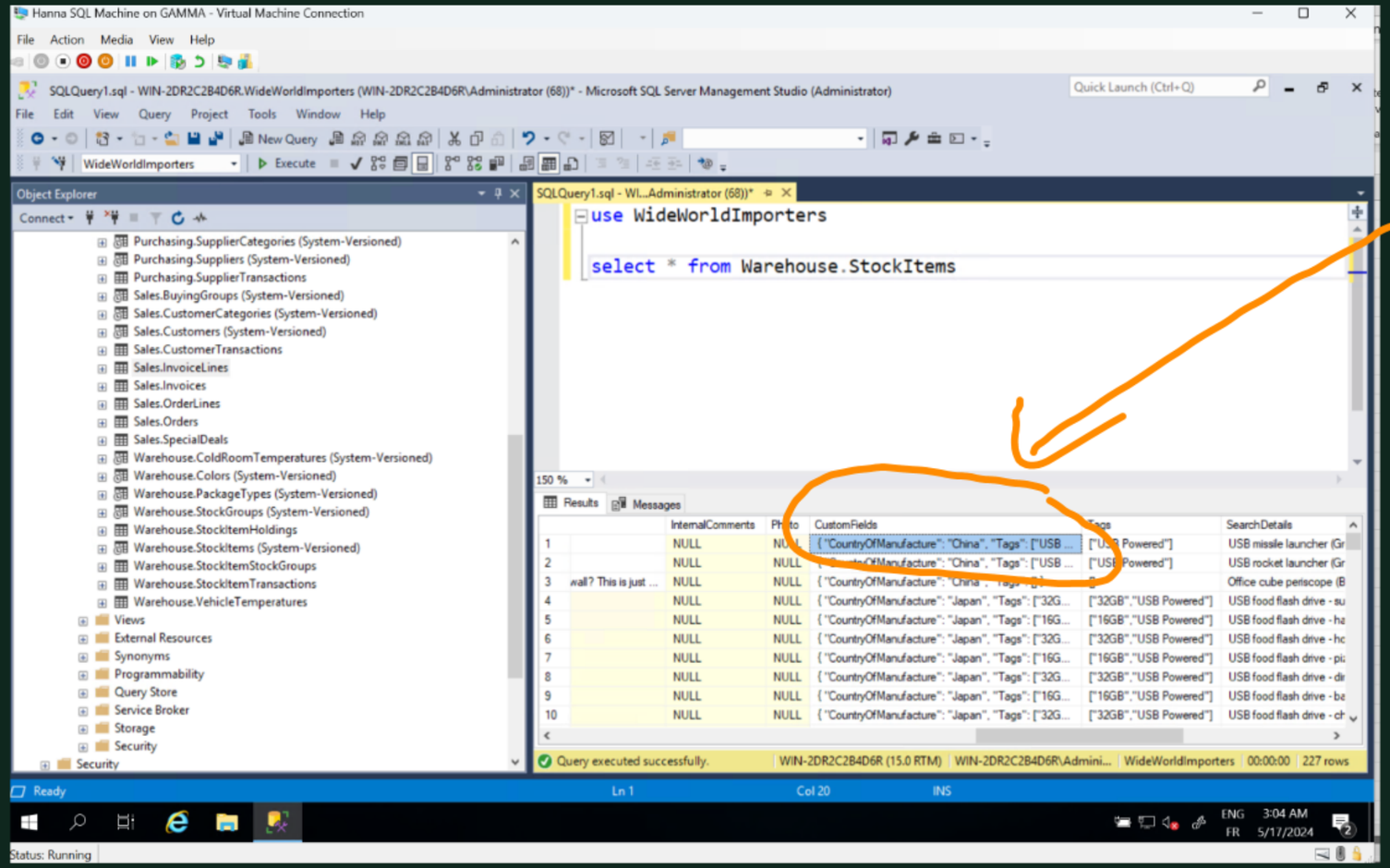
NoT C → D
NoT D → C

BCNF:

NoT C → A
NoT C → B
NoT D → A
NoT D → B

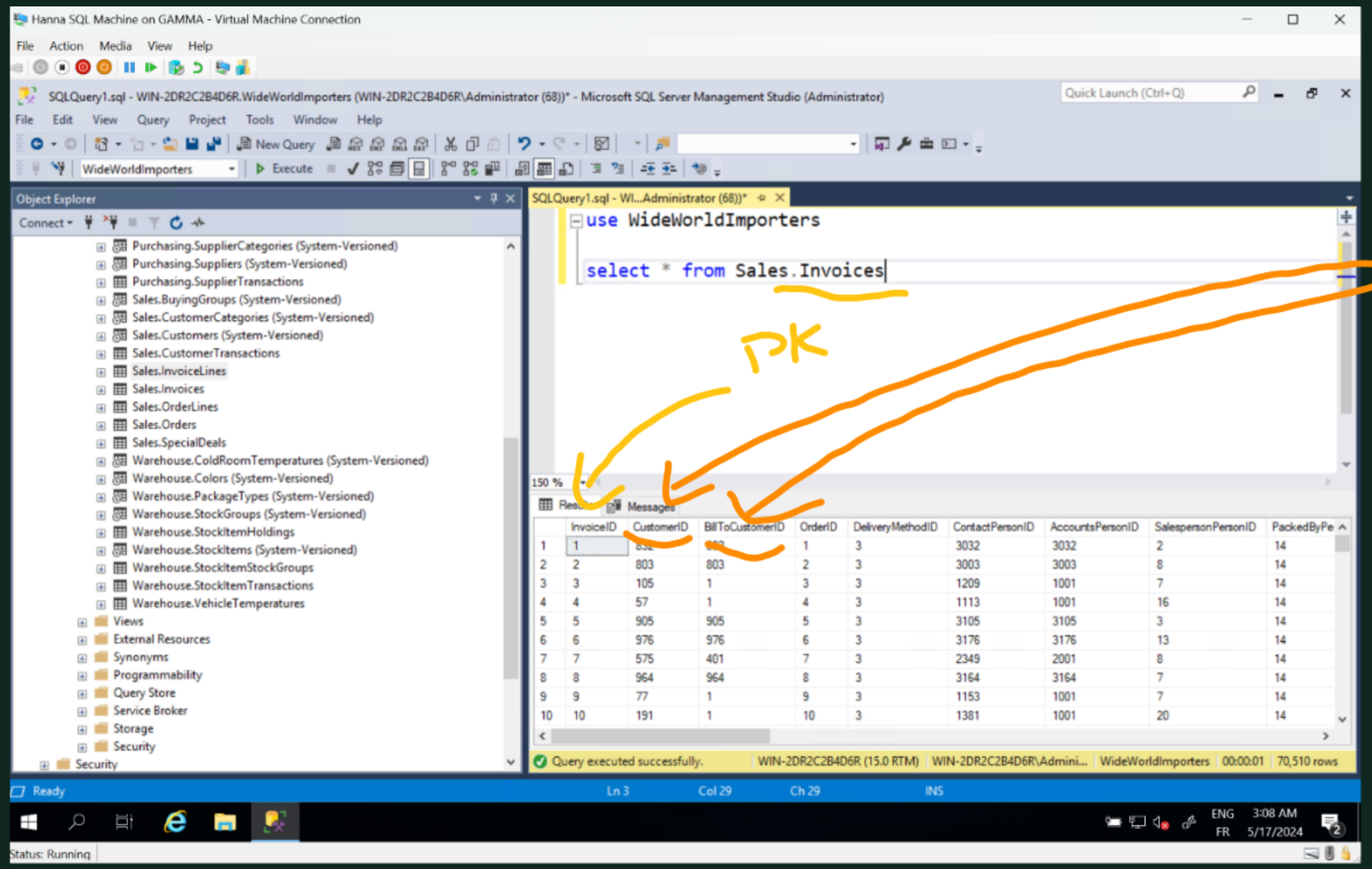
(+fr. completeness: NoT A → B, NoT B → A)

WWI example : 1NF



{ "Country" : China, "Tags" : [USB, Green] Breach 1NF, yo }

WWI example : 3NF



non-key attributes (related in CUSTOMERS table)

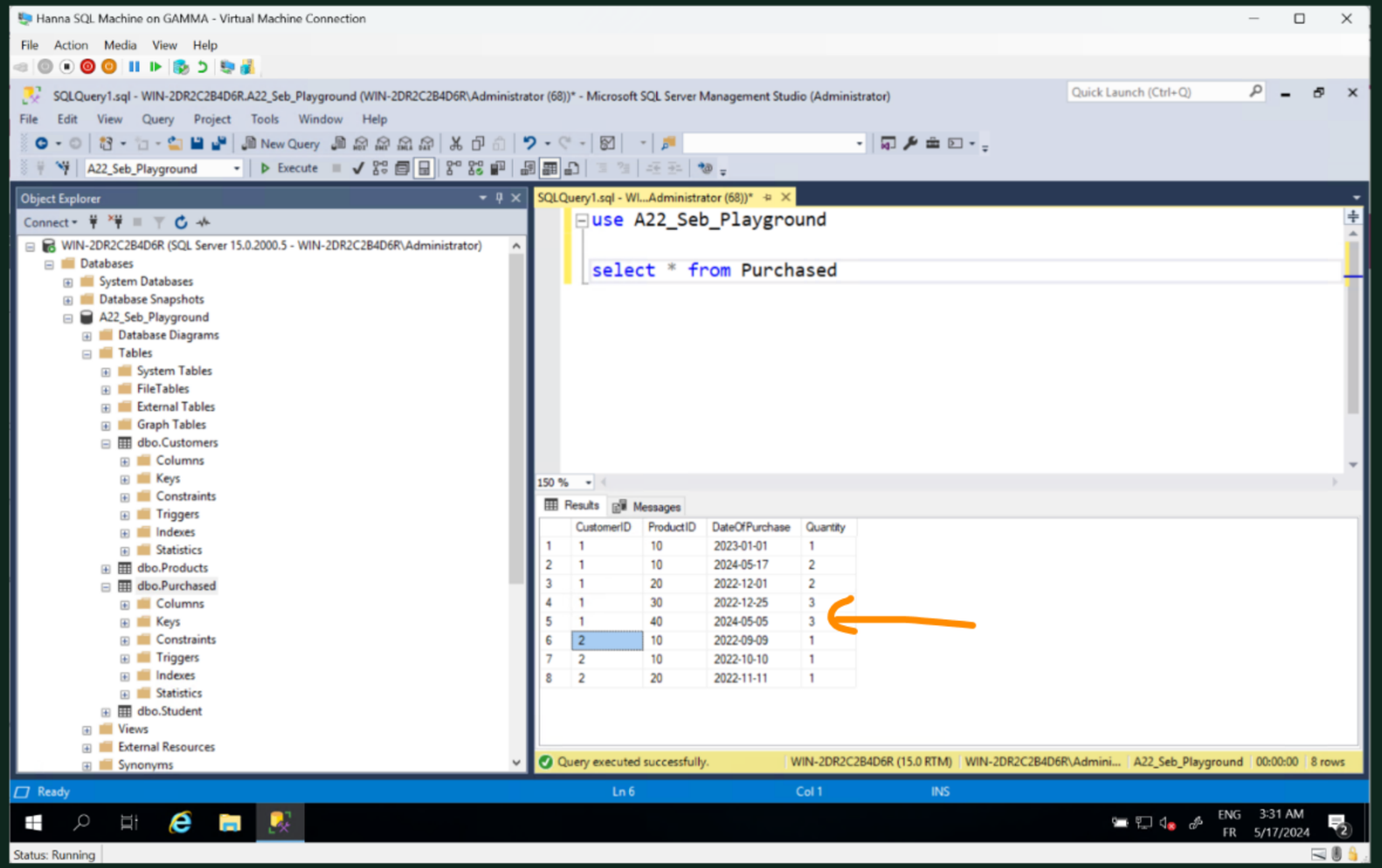
Breach of 3NF!

Lunch

Back at 14:00

Seb's playground DB

1. He "Purchased" table, add the line:



1	40	2024-05-05	3

Group by Query (1)

Result:

City	Count of Customers
Nice	2
Paris	1

Group by Query (2)

Dimensionality Reduction

ID	FName	LName	City
1	Sebastien	C	Nice
2	Jennifer	H	Nice
3	Vincent	P	Paris

[3 x 4]

→

City	Count
Nice	2
Paris	1

→ [2 x 2]

E/R Diagram

PRODUCTS B

CUSTOMERS A

ProductID	Description	UnitPrice
10	Laptop	800
20	Desktop	1000
30	Tablet	600
40	Flat Monitor	150

CustomerID	FirstName	LastName	Email	City
1	Seb	C	sebastien@dst	Nice
2	Jennifer	H	jennifer@dsti.c	Nice
4	Vincent	P	vincent@dsti.c	Paris

(0,n)

(0,n)

CustomerID	ProductID	DateOfPurchase	Quantity
1	10	1/1/2023	1
1	10	5/17/2024	2
1	20	12/1/2022	2
1	30	12/25/2022	3
1	40	5/5/2024	3
2	10	9/9/2022	1
2	10	10/10/2022	1
2	20	11/11/2022	1

PURCHASED AB

Division Query (1)

→ Simplest, most effective & optimized way to solve the

question:

Give all entityA (Customers) that relationship AB (Purchased) all entity B ? (PRODUCTS)

Division Query (2)

Conditions to fulfill:

- Entity A
- Entity B
- Relationship AB binding them

Division Query (3)

Skeleton:

- Outer query (entity A)
- Inner (sub) query (entity B)
- Inner-inner (sub-sub) query (Relationship AB)

Division Query (4)

Why does it work?

I am happy ⟶

I am not happy ⟶

I am not not happy ⟶

All Customers & All Products

PRODUCT

ProdID	Desc
10	laptop
20	Monitor
30	PC
40	Batteries

(2) count = 4

PURCHASED

CustID	ProdID
1	10
2	20
1	30
1	40
2	10
1	20

(3) (1) count = 4

1 | (4) DISTINCT

2 | 1

Break

Back at 15:20

Programmatic SQL (1)

What's the problem with "regular" SQL?

weakness / limitation

Programmatic SQL (2)

Aim: bridge gap between SQL and functional languages (e.g., Python)

Programmatic SQL (3)

Why?

1. Locality of data
2. Intellectual Property

Programmatic SQL (4)

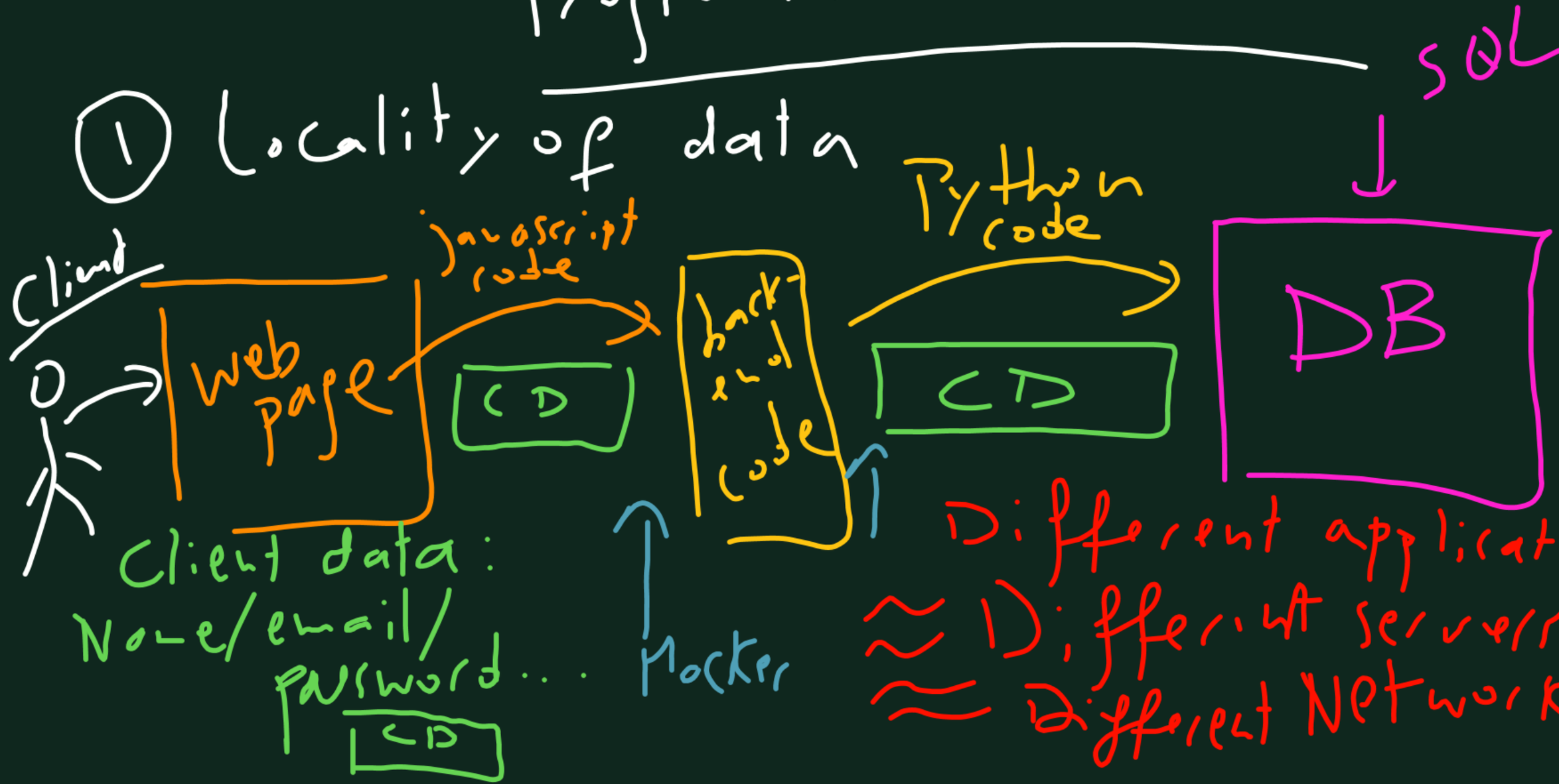
1. Locality of data

Generally, data flows in 3 "parts"

1. UI (website) Front-end
2. Back-end
3. DB

Programmatic SQL (5)

① Locality of data



Programmatic SQL (6)

(2) Intellectual Property

→ "He who owns the code, owns the data".

Programmatic SQL (7)

Benefits: (loop, define variables, set conditions, ...)

- Add functional behavior to SQL
- BEGIN, END, IF, RETURN, ELSE

Programmatic SQL (8)

→ Very vendor-specific

→ Microsoft has T-SQL

(Transact-SQL)

Programmatic SQL (9)

Python

```
x = 2
```

1. x is an int
2. x has value 2

T-SQL

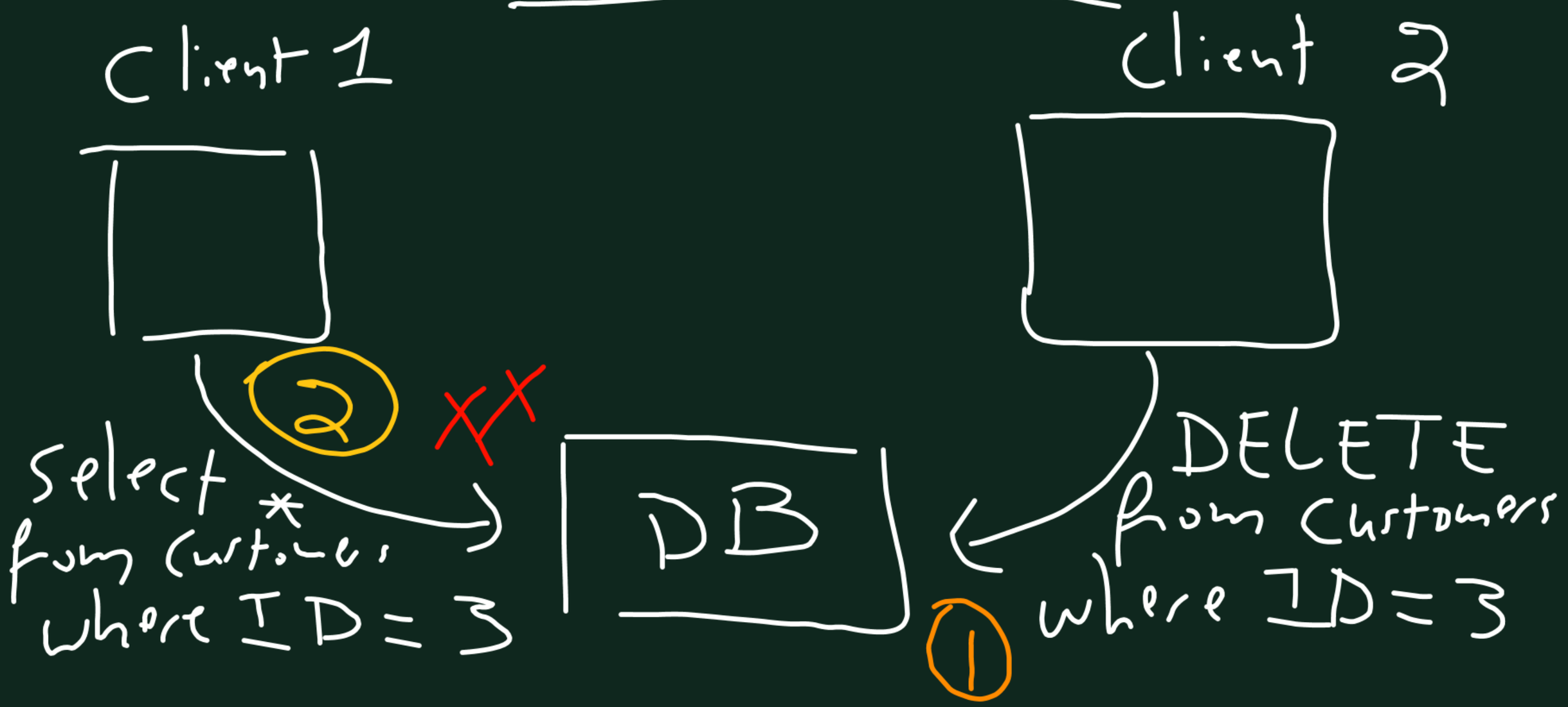
```
DEFINE @x int;
SET @x = 2;
```

Programmatic SQL (10)

Exciting and new things:

- Transactions
- Stored Procedures
- Triggers

Transactions (1)



Transactions (2)

By nature, RDMS are:

- Multi-user
- Concurrent
- Networked

Transactions (3)

Goal: "Freeze" order of actions until COMMIT or ROLLBACK

Break

Back at 16:30

Stored Procedures

- Reusable pieces of code
- Closest thing to actual functions in programming languages

Reusable parametrizable

Triggers

Mechanism that executes a stored procedure based on actions

fires

events

CREATE
UPDATE
DELETE

Application: YOU! (1)

Pre-admission

Fname
Lname
Email
City
Application Status
Student Number

1 = Application Status
NULL = Student Number

Admission

UPDATE

Application Status = 2
(trigger)

Student Number
(Stored procedure)

Application: YOU! (2)

Applicant	Student
Applicant 1 ✓	1
Applicant 2 ✗	✗
Applicant 3 ✓	2

Student Number

2024 0001

4 4

2024 0002